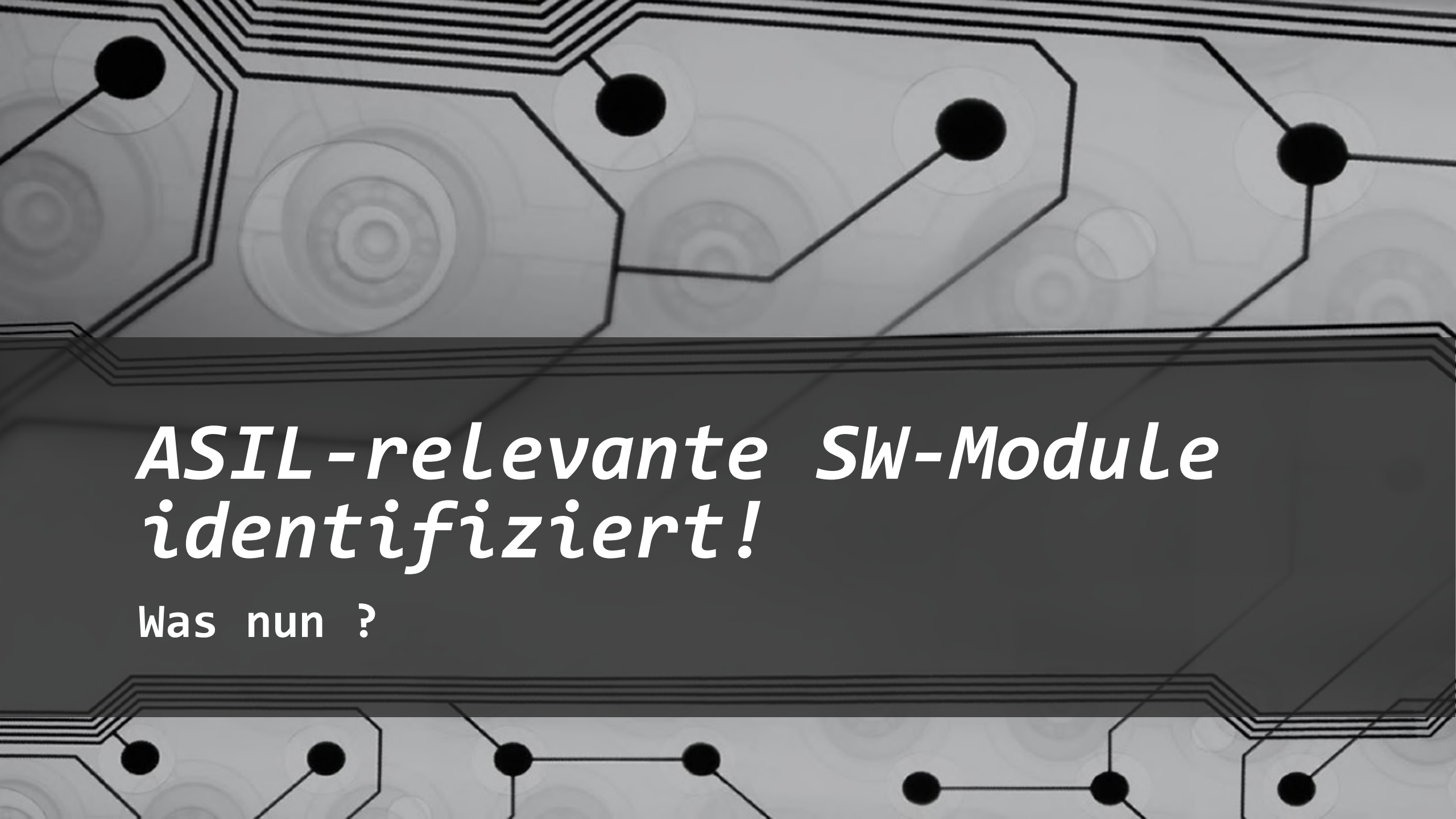

ASIL-relevante SW-Module identifiziert!
Was nun ?

ASIL-relevante SW-Module testen

- Blick in die EN 26262
- Häufige Irrtümer in der Praxis
- Funktionale Tests in die Tiefe
- Funktionale Tests weiter optimieren
- Ein Beispiel als Beispiel
- Zusammenfassung

Blick in die EN 26262
Testüberdeckung - Modultest

Um die Vollständigkeit von Testfällen sicherzustellen, verlangt die Norm ISO 26262 die Messung der strukturellen Testabdeckung (White-Box-Test) (siehe Punkt 8.4.5 der Norm 26262-6)

Testabdeckungsstufe	ASIL A	ASIL B	ASIL C	ASIL D
Anweisungsüberdeckung (C0)	++	++	+	+
Zweigüberdeckung (C1)	+	++	++	++
MC/DC	+	+	+	++

++ steht für "sehr zu empfehlen (highly recommended)"
+ steht für "zu empfehlen (recommended)"

Blick in die EN 26262
Testüberdeckung - Gesamttest

Außerdem verlangt die ISO 26262 Funktionsüberdeckung. Hierfür muss jeder Funktionsaufruf einmal abgedeckt sein. (Siehe Punkt 10.4.6 der Norm 26262-6)

Methoden		ASIL A	ASIL B	ASIL C	ASIL D
1a	Funktionsüberdeckung	++	++	+	+
1b	Aufrufüberdeckung	+	++	++	++

Häufige Irrtümer in der Praxis

1. **Funktionale Testfälle und Codeabdeckungs-Testfälle können getrennt voneinander erstellt werden!**

Häufige Irrtümer in der Praxis

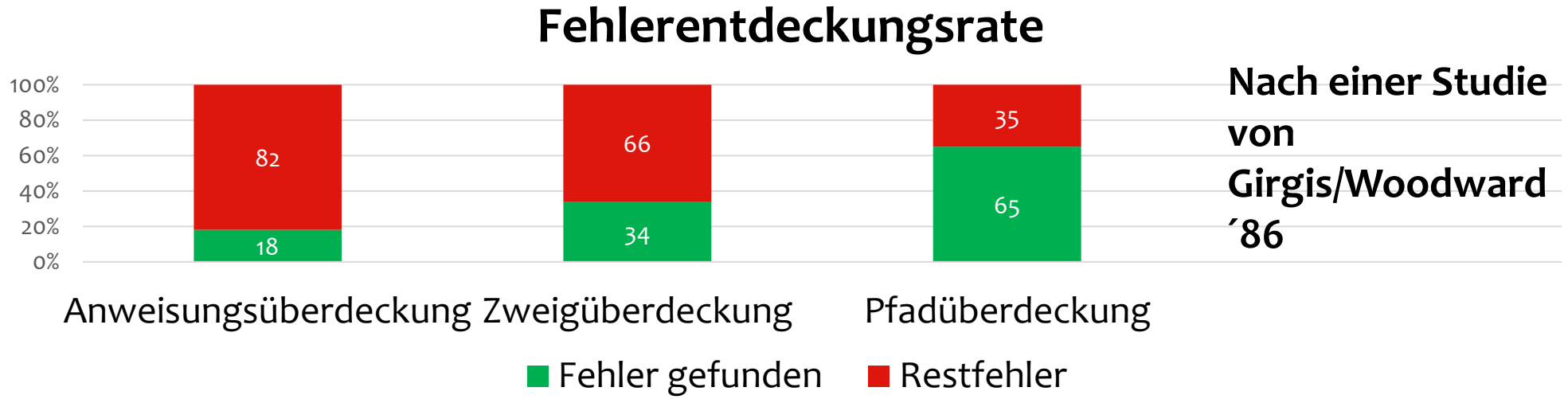
1. **Funktionale Testfälle und Codeabdeckungs-Testfälle können getrennt voneinander erstellt werden!**

   (vermeintliche) Vorteile:

   - Die SW-Qualität wird erhöht, weil jetzt mit zwei unterschiedlichen Testmethoden nach Fehlern gesucht wird und sich die Anzahl der Testfälle erhöht!
   - Vorhandene Teststrategie (z.B. Test gegen Anforderungen) können unverändert beibehalten werden.
   - Zusätzliche Testaufgaben (Codeüberdeckung) können parallel von einem anderen Team erstellt werden (z.B. Auftrag extern vergeben)
   - Die Vorgaben der Normen werden erfüllt.

Häufige Irrtümer in der Praxis

- **Die SW-Qualität wird erhöht, weil jetzt mit zwei unterschiedlichen Testmethoden nach Fehler gesucht wird und sich die Anzahl der Testfälle erhöht!**



Nach einer Studie von Girgis/Woodward ´86

Korrekte Anwendung in der Praxis

FALSCH: Funktionale Testfälle und Codeabdeckungs-Testfälle können getrennt voneinander erstellt werden!

RICHTIG: Funktionale Testfälle und Codeabdeckungs-Testfälle gemeinsam erstellen!

Reihenfolge:

1. Funktionale Testfälle, z.B. Test gegen Anforderungen.
2. Code-Coverage messen.
3. Mit Whitebox-Tests zusätzliche Testfälle finden um die Lücken in den funktionalen Testfällen (nicht ausgeführte Anweisungen oder nicht erreichte Verzweigungen) zu schließen.

Korrekte Anwendung in der Praxis

1. **Funktionale Testfälle und Codeabdeckungs-Testfälle gemeinsam erstellen!**

Vorteile:

- Die SW-Qualität wird erhöht, weil mit Hilfe der Whitebox-Tests Lücken in den funktionalen Testfällen erkannt und beseitigt werden können.
- Die Vorgaben der Normen werden **korrekt** erfüllt.

Korrekte Anwendung in der Praxis

1. **Funktionale Testfälle und Codeabdeckungs-Testfälle gemeinsam erstellen!**

   Nachteile:

   - Vorhandene Teststrategien müssen angepasst werden (Möglichkeit zur Coverage-Messung)
   - Coverage-Testtools verändern den Code (+ zus. Laufzeit, + zus. Speicherbedarf)
   - Testdurchführung findet **NICHT** mit dem Orginalcode statt!

   Praxistipp:

   Regressionstest mit Orginalcode durchführen!

Häufige Irrtümer in der Praxis

1. **Funktionale Testfälle und Codeabdeckungs-Testfälle können getrennt voneinander erstellt werden!** F
2. **Durch 100% Codeabdeckung bei der Erstellung von Whitebox-Testfällen wird die SW-Qualität deutlich erhöht, da sehr gründlich getestet wird.** F

FALSCH: (Durch die Studien der reinen Coverage-Tests eindeutig widerlegt)!

Korrekte Anwendung in der Praxis

RICHTIG: Die Code-Coverage misst nicht die Qualität des Codes sondern die Qualität der funktionalen Testfälle!

Als Ergänzung der funktionalen Tests erhöht die Whitebox-Testmethode die Effektivität der funktionalen Testfälle und damit die SW-Qualität!

Praxistipp:

Die Whitebox-Testmethode ist in der Praxis hauptsächlich nur auf Modultest-Ebene sinnvoll einsetzbar, da zum Erreichen der Code-Coverage oftmals Funktionen durch Platzhalter (Stubfunktionen) ersetzt werden müssen, um effizient in bestimmte Programmzweige zu gelangen.

Häufige Irrtümer in der Praxis

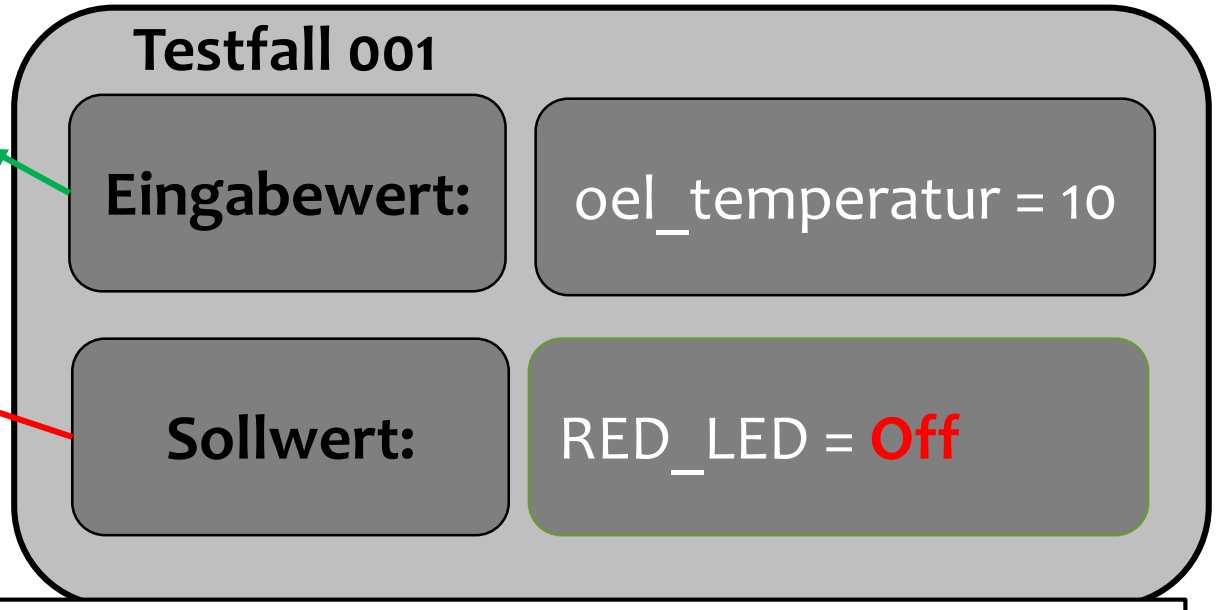
1. **Funktionale Testfälle und Codeabdeckungs-Testfälle können getrennt voneinander erstellt werden!** F
2. **Durch 100% Codeabdeckung bei der Erstellung von Whitebox-Testfällen wird die SW-Qualität deutlich erhöht, da sehr gründlich getestet wird.** F
3. **Für die Testfall-Erstellung der Codeabdeckungs-Testfälle wird nur der Quellcode benötigt!**

Häufige Irrtümer in der Praxis

3. **Für die Testfall-Erstellung der Codeabdeckungs-Testfälle wird nur der Quellcode benötigt!**

RICHTIG: Eingabewert aus dem Sourcecode ermitteln

```
if (oel_temperatur <= 50 )
{
    setRED_LED(Off);
}
else
{
    setRED_LED(On);
}
```



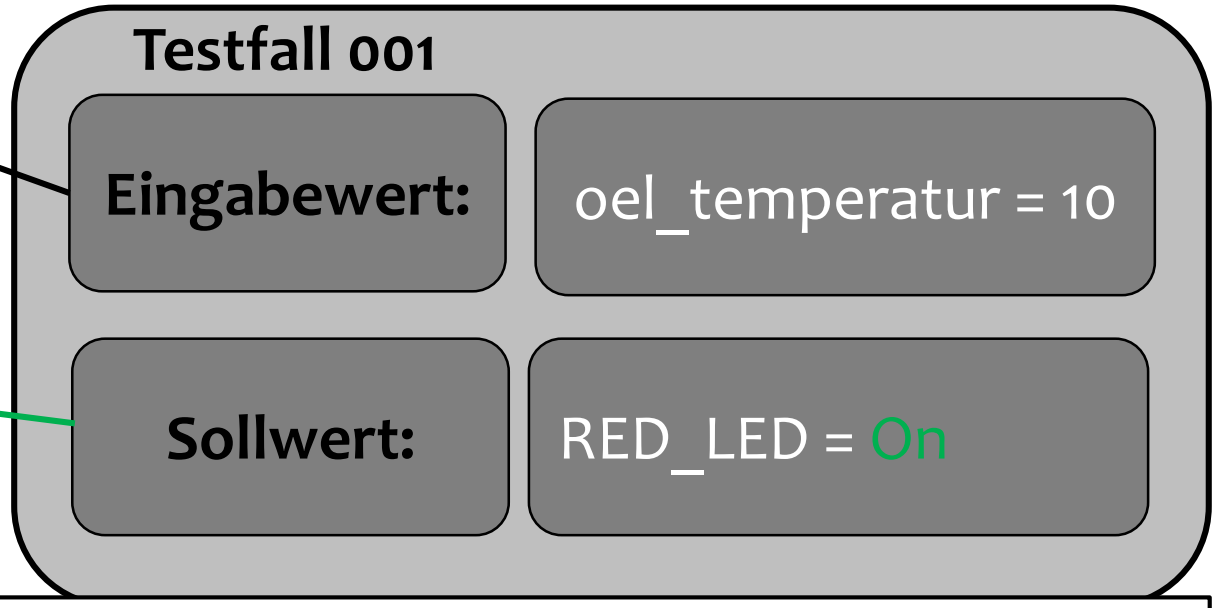
FALSCH!: Sollwerte aus dem Sourcecode ermitteln

Korrekte Anwendung in der Praxis

3. Für die Testfall-Erstellung der Codeabdeckungs-Testfälle wird zur Ermittlung der Eingabeparameter der Quellcode benötigt. **Zur Ermittlung der Sollwerte muss die Spezifikation verwendet werden!**

```
if (oel_temperatur <= 50 )
{
    setRED_LED(Off);
}…
```

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

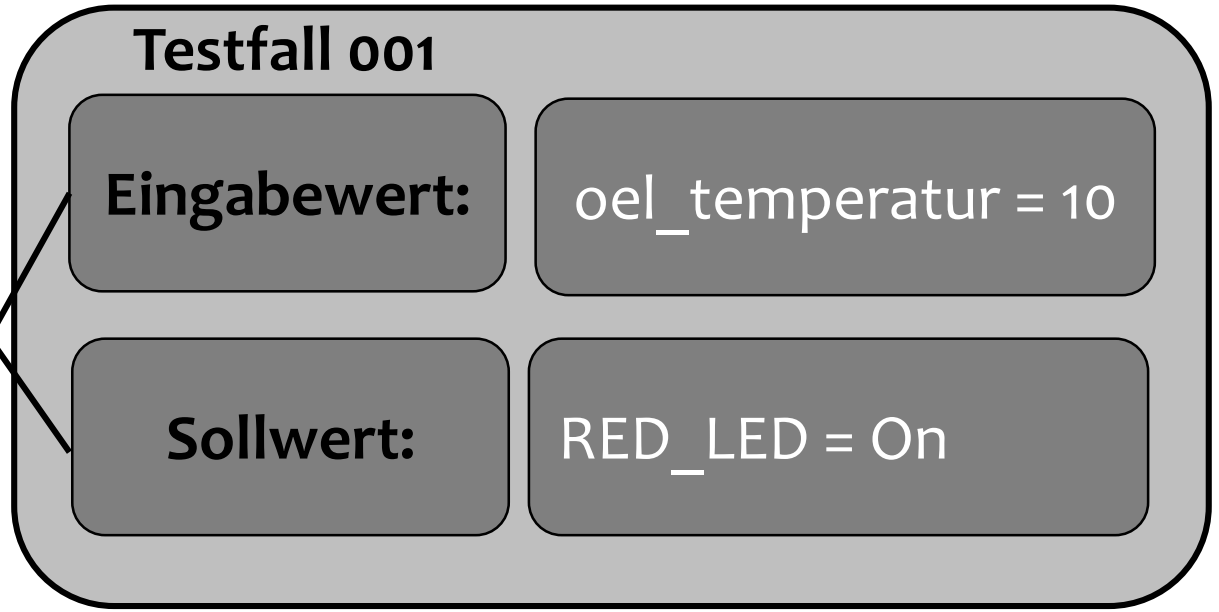


RICHTIG: Sollwerte aus der Spezifikation ermitteln!

Korrekte Anwendung in der Praxis

Praxistipp:
Durch gute Sourcecode-Dokumentation (z.B. mit DOXYGEN) werden Spezifikationen und Code an einer Stelle verwaltet!

```
// ------------------------------
// ! REQ_ID 4711
// ! Bei einer Öltemperatur von
// ! 50°C oder weniger, muss die
// ! rote Kontroll-LED leuchten
// ------------------------------
if (oel_temperatur <= 50 )
{
   setRED_LED(Off);
} …
```



Häufige Irrtümer in der Praxis

1. Funktionale Testfälle und Codeabdeckungs-Testfälle können getrennt voneinander erstellt werden! F
2. Durch 100% Codeabdeckung bei der Erstellung von Whitebox-Testfällen wird die SW-Qualität deutlich erhöht, da sehr gründlich getestet wird. F
3. Für die Testfall-Erstellung der Codeabdeckungs-Testfälle wird nur der Quellcode benötigt! F
4. Man kann immer 100% Codeabdeckung erreichen

Häufige Irrtümer in der Praxis

4. Man kann immer 100% Codeabdeckung erreichen

```
const int divisor = 100;
…
if ( divisor UNGLEICH 0 )
{
   quotient = wert GETEILT divisor;
}
else
{
   print („ERROR:Division durch 0!“);
}
```

Variable ‚divisor‘ wird als Konstante mit einem festen Wert vorbelegt.

Programmier-Richtlinien verlangen eine Prüfung von Divisoren auf UNGLEICH NULL!

Nicht erreichbare Anweisung. 100% C0 und C1-Abdeckung nicht erreichbar!

Korrekte Anwendung in der Praxis

4. Man kann nicht immer 100% Codeabdeckung erreichen

Zusätzliche Maßnahmen ergreifen, wenn 100% Codeüberdeckung nicht erreichbar ist.

Praxistipp:

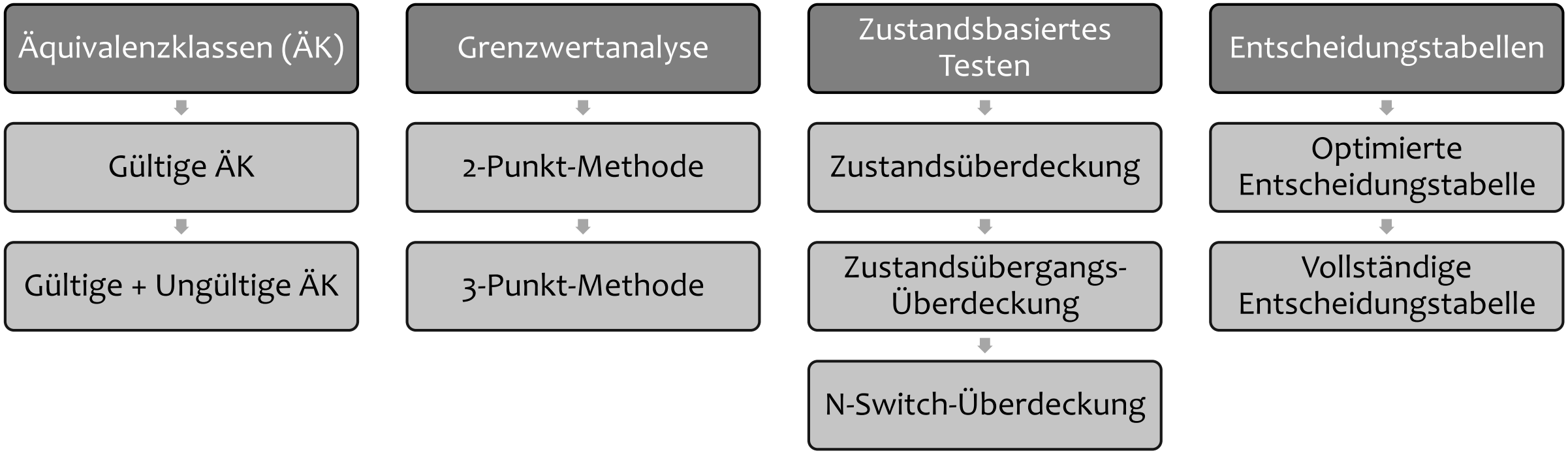
- Absicherung durch Review
- Umstellung des Programmcodes
- Eventuell: Entfernen nicht erreichbarer Code-Bereiche

Häufige Irrtümer in der Praxis

1. Funktionale Testfälle und Codeabdeckungs-Testfälle können getrennt voneinander erstellt werden. F
2. Durch 100% Testabdeckung bei der Erstellung von Codeabdeckungs-Testfällen wird die SW-Qualität deutlich erhöht, da sehr gründlich getestet wird. F
3. Für die Testfall-Erstellung der Codeabdeckungs-Testfälle wird nur der Quellcode benötigt. F
4. Man kann immer 100% Codeabdeckung erreichen. F

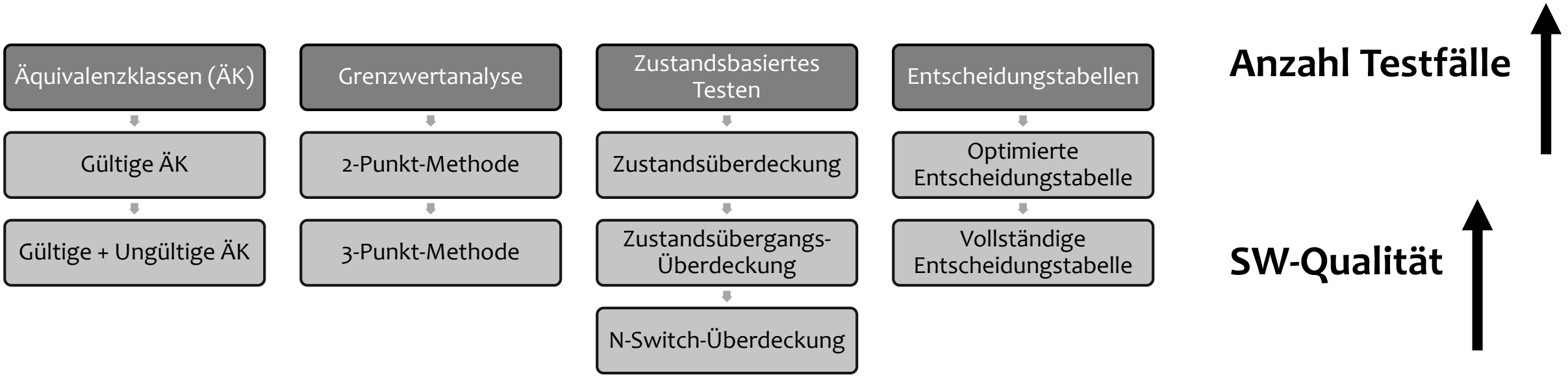
Funktionale Tests in die Tiefe

- Blackbox-Testmethoden -

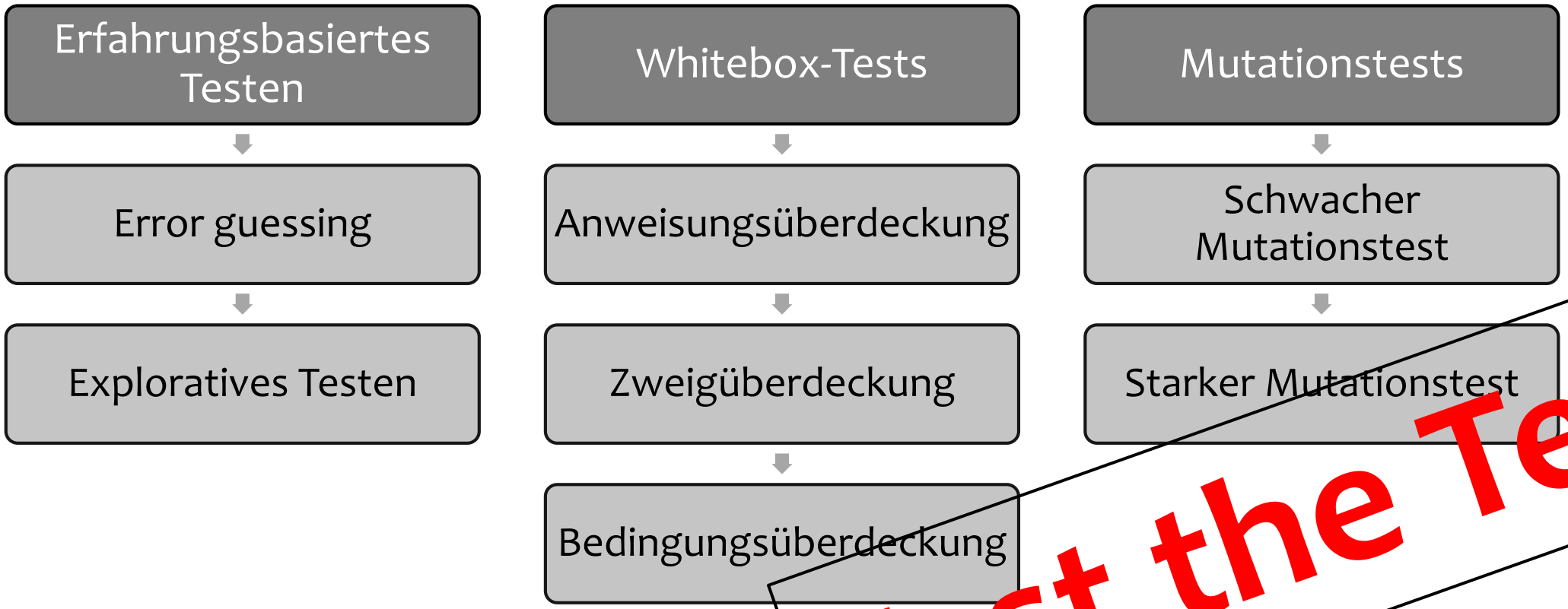


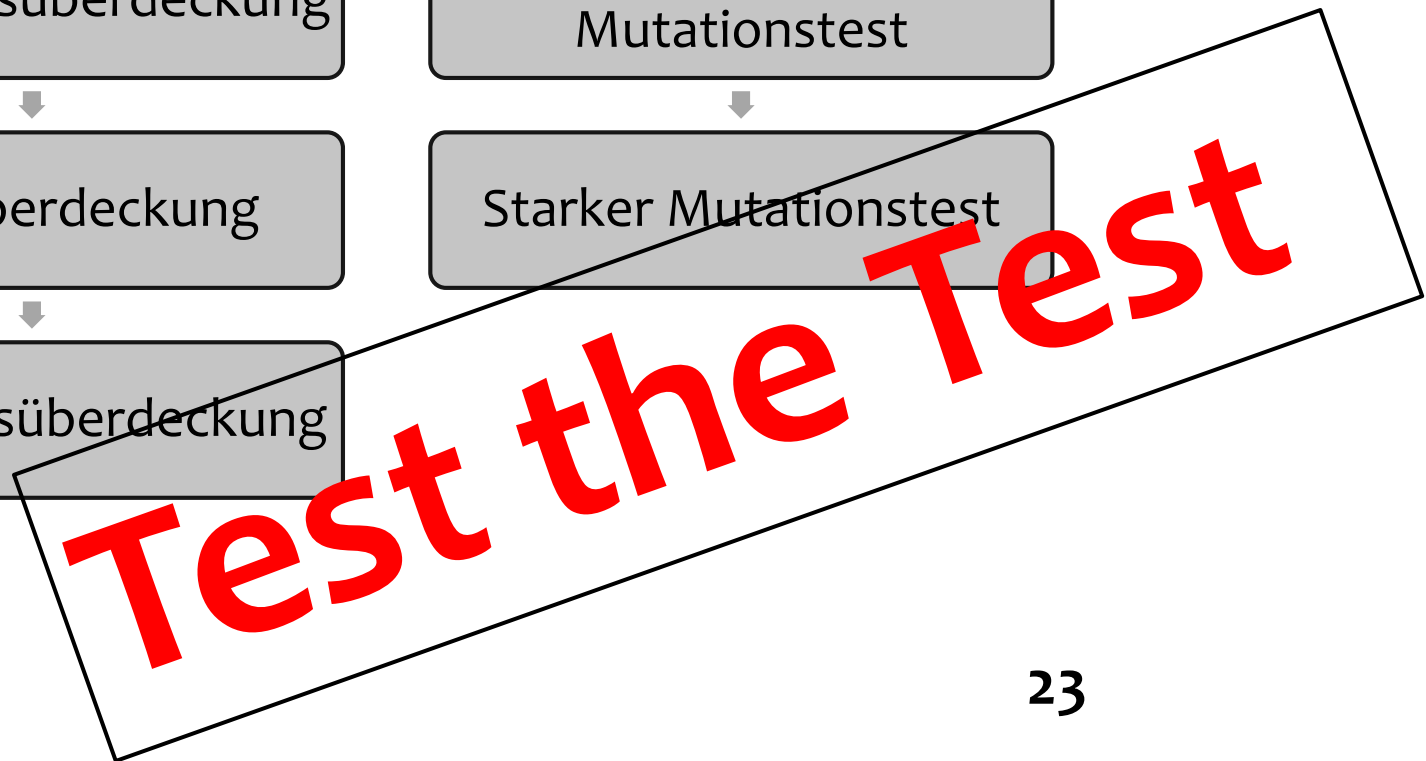
Funktionale Tests in die Tiefe

- Blackbox-Testmethoden -



Funktionale Tests weiter optimieren



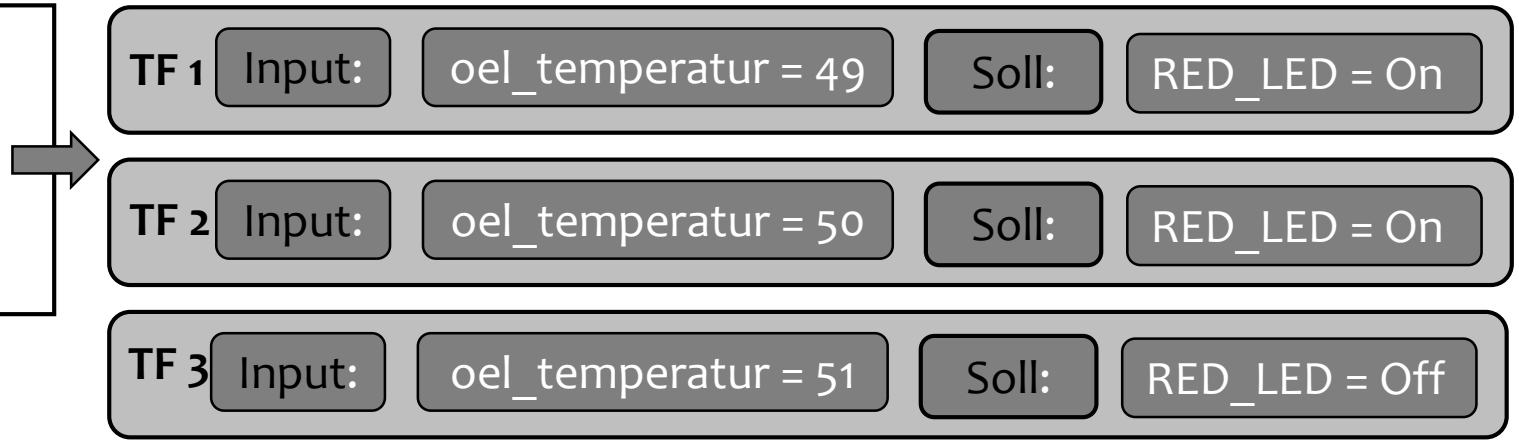


Ein Beispiel als Beispiel

- Funktionale Testfälle erstellen (Blackbox-Tests)

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

Testfälle für die Grenzwertanalyse



Testdurchführung

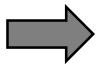
- TF 1: failed [Soll: RED_LED = On, Ist: RED_LED = Off]
- TF 2: passed [Soll: RED_LED = On, Ist: RED_LED = On]
- TF 3: failed [Soll: RED_LED = Off, Ist: RED_LED = On]

Ein Beispiel als Beispiel

- Fehlersuche und Fehlerkorrektur nach Blackbox-Tests

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

```
// !Prüfen der Öltemperatur
if (oel_temperatur < 50 )
{
   setRED_LED(Off);
}
else
{
   setRED_LED(On);
}
```



```
// !Prüfen der Öltemperatur
if (oel_temperatur <= 50 )
{
   setRED_LED(On);
}
else
{
   setRED_LED(Off);
}
```

Ein Beispiel als Beispiel

- Fehlernachtest durchführen

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

```
// !Prüfen der Öltemperatur
if (oel_temperatur <= 50 )
{
   setRED_LED(On);
}
else
{
   setRED_LED(Off);
}
```

Testfälle für die Grenzwertanalyse

TF 1	Input:	oel_temperatur = 49	Soll:	RED_LED = On
TF 2	Input:	oel_temperatur = 50	Soll:	RED_LED = On
TF 3	Input:	oel_temperatur = 51	Soll:	RED_LED = Off

Testdurchführung

- TF 1: passed
- TF 2: passed
- TF 3: passed

Ein Beispiel als Beispiel

- Code-Coverage ermitteln

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

```
// !Prüfen der Öltemperatur
if (oel_temperatur <= 50 )
{
   setRED_LED(On);
}
else
{
   setRED_LED(Off);
}
```

Testfälle für die Grenzwertanalyse

TF 1	Input:	oel_temperatur = 49	Soll:	RED_LED = On
TF 2	Input:	oel_temperatur = 50	Soll:	RED_LED = On
TF 3	Input:	oel_temperatur = 51	Soll:	RED_LED = Off

Testdurchführung:

- TF 1: passed
- TF 2: passed
- TF 3: passed

Code-Coverage:

- Anweisungsüberdeckung: 100%
- Zweigüberdeckung: 100%

Ein Beispiel als Beispiel

- Nur Whitebox-Tests durchführen

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

```
// !Prüfen der Öltemperatur
if (oel_temperatur < 50 )
{
   setRED_LED(Off);
}
else
{
   setRED_LED(On);
}
```

Testfälle für die Zweigüberdeckung (C1)

TF 1	Input:	oel_temperatur = 10	Soll:	RED_LED = On
TF 2	Input:	oel_temperatur = 75	Soll:	RED_LED = Off

Testdurchführung:

- TF 1: failed
- TF 2: failed

Code-Coverage:

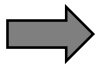
- Zweigüberdeckung: 100%
- Anweisungsüberdeckung: 100%

Ein Beispiel als Beispiel

- Fehlersuche und Fehlerkorrektur nach Whitebox-Tests

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

```
// !Prüfen der Öltemperatur
if (oel_temperatur < 50 )
{
   setRED_LED(Off);
}
else
{
   setRED_LED(On);
}
```



```
// !Prüfen der Öltemperatur
if (oel_temperatur < 50 )
{
   setRED_LED(On);
}
else
{
   setRED_LED(Off);
}
```

Ein Beispiel als Beispiel

- Fehlernachtest durchführen

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

```
// !Prüfen der Öltemperatur
if (oel_temperatur < 50 )
{
   setRED_LED(On);
}
else
{
   setRED_LED(Off);
}
```

Testfälle für die Zweigüberdeckung (C1)

TF 1	Input:	oel_temperatur = 10	Soll:	RED_LED = On
TF 2	Input:	oel_temperatur = 75	Soll:	RED_LED = Off

Testdurchführung:

- TF 1: passed
- TF 2: passed

Code-Coverage:

- Zweigüberdeckung: 100%
- Anweisungsüberdeckung: 100%

Ein Beispiel als Beispiel

- nach Whitebox-Tests

Anforderung 4711:
Bei einer Öltemperatur von 50°C oder weniger, muss die rote Kontroll-LED leuchten.

Testfälle für die Zweigüberdeckung (C1)

TF 1	Input:	oel_temperatur = 10	Soll:	RED_LED = On
TF 2	Input:	oel_temperatur = 75	Soll:	RED_LED = Off

```
// !Prüfen der Öltemperatur
if (oel_temperatur < 50 )
{
   setRED_LED(On);
}
else
{
   setRED_LED(Off);
}
```

Testdurchführung:

- TF 1: passed
- TF 2: passed

Zusammenfassung

- **Blackbox-Tests testen die Funktionen, Whitebox-Tests testen die Qualität der Blackbox-Tests!**
- **Funktionale Tests und Whitebox-Tests in Kombination, führen zu einer Verbesserung der SW-Qualität, Whitebox-Tests alleine dagegen nicht!**
- **Auch für Whitebox-Tests wird zur Erstellung der Testfälle die Spezifikation benötigt!**

Zusammenfassung

- **100% Abdeckungsgrad bei Whitebox-Tests ist nicht immer erreichbar.**
- **Eine Erhöhung der Testtiefe führt zu einer verbesserten Testabdeckung aber auch zu mehr Testfällen.**
- **Durch Maßnahmen wie z.B. Erfahrungsbasiertes Testen, Whitebox-Tests, Mutationstests kann man die Qualität der Funktionalen Tests prüfen, bzw. weiter verbessern.**

I B
E

Informatikbüro
Erlenbach

Danke für Ihre Aufmerksamkeit!

Noch Fragen?